DualDB Memory Plugin 架构说明文档

项目名称: DualDB Memory Plugin

作者 : 孙旭

联系邮箱: yzxy6151@gmail.com

创建时间: 2025-07-22

用途 : 本文档用于正式归档 DualDB Memory Plugin 架构设计，明确原创性结构主张，供后续署名、学术提交或项目推广使用。本阶段暂不公开源代码。

版本：v1.0　用途：结构原型归档

引言 / Introduction

DualDB Memory Plugin 是一个轻量级 AI 对话记忆管理组件，旨在模拟人类记忆的短期与长期分离结构，提供可自动轮换、摘要归档、多存储后端的插件化设计。其核心架构通过双数据库（Active + Archive）结合触发机制与可插拔摘要器实现长期上下文保存，支持未来扩展至 RocksDB、大模型摘要等复杂应用场景。

该插件的设计思路独立完成，未借助主流框架或已有系统模板，具有结构清晰、功能明确、可模块扩展的特性，适合独立归档并作为原创结构主张基础。

第 1 章：架构概览

DualDB Memory Plugin 旨在模拟人类记忆系统中的“短期记忆”与“长期记忆”机制。它通过两个独立的存储结构：

- Active（短期库存）：快速写入与读取，保存最新对话。
- Archive（长期库存）：存储已摘要压缩的历史记录，保持长期上下文。

当 Active 区达到触发条件（如消息条数阈值、关键词或时间间隔）时，系统会：

1. 生成摘要：调用可插拔的摘要接口压缩 Active 中的历史。
2. 写入 Archive：将摘要结果持久化到 Archive 存储。
3. 清空 Active：删除已归档的短期数据，为新对话腾出空间。

该设计封装了以下循环：

- 向 Active 中追加新对话。
- 在满足阈值、关键词或时间条件时触发轮回。
- 对历史记录进行摘要并写入 Archive。
- 清理 Active，保持存储增长的可控性。

这样可确保插件在不失去关键信息的前提下，持续保留对话上下文，同时提供简洁且强大的接口。

第 2 章：模块组成

DualDB Memory Plugin 的核心模块划分如下：

1. **DualDBManager**
   - 统一接口，负责协调 Active 和 Archive 存储，以及触发轮回摘要流程。
   - 提供 append(user, message)、get_context()、force_rotate() 等方法。
2. **存储模块 / Storage Backends**
   - **ActiveStore**：高吞吐写入模块，支持 JSON 或 SQLite（后续可扩展 RocksDB）。
   - **ArchiveStore**：摘要后历史存储模块，长期保存关键上下文。
   - 二者通过统一的存储接口 (StoreInterface) 实现替换与扩展。
3. **摘要器模块 / Summarizer Interface**
   - 定义 BaseSummarizer 抽象类或接口，保证一致调用。
   - 默认提供 StubSummarizer（测试用）和 OpenAISummarizer（示例）。
   - 开发者可自行实现本地模型摘要或第三方 API 接入。
4. **触发器 / Trigger Mechanisms**
   - 基于消息 **数量阈值**（threshold）触发。
   - 基于 **关键词匹配**（keywords list）触发。
   - 基于 **时间间隔**（time delta）定时触发。
   - 支持多条件组合，实现灵活轮回策略。

第 3 章：轮换机制

3.1 触发条件 / Trigger Conditions

DualDB Memory Plugin 提供三种可配置的轮换触发条件:

- **消息数阈值（Threshold）**：当 Active 中的消息条目数达到预设阈值时触发。
- **关键词触发（Keywords）**：当新消息包含任一指定关键词，触发摘要归档。
- **定时触发（Time Delta）**：基于周期性时间间隔自动执行轮换，例如每隔 1 小时。

插件可同时启用多种触发策略，并根据配置逻辑（例如“任一满足”或“全部满足”）决定触发时机。

3.2 轮换流程 / Rotation Process

1. **检测触发**：在 append() 或后台定时任务中检查触发条件。
2. **锁定 Active**：为保证一致性，暂时锁定 ActiveStore，阻止新写入。
3. **摘要生成**：调用 Summarizer.summarize(active_messages)，生成压缩后的摘要文本。
4. **写入 Archive**：将摘要文本与关联元数据（时间戳、重要性标记）写入 ArchiveStore。
5. **清理 Active**：删除已归档的旧消息，使 ActiveStore 回到空状态或剩余最新未归档消息。
6. **释放锁**：完成后解除锁定，恢复正常写入。

第 4 章：摘要接口

4.1 接口定义 / Interface Definition

DualDB Memory Plugin 通过 BaseSummarizer 抽象类或接口，定义以下核心方法:

```
class BaseSummarizer:
  def summarize(self, messages: List[Message]) -> str:
    """
    将传入的消息列表压缩为简洁摘要。
    :param messages: 要摘要的消息列表（含用户角色、时间戳等）
    :return: 提取或生成的摘要文本
    """
```

4.2 默认实现 / Default Implementations

1. **StubSummarizer**：提供最简占位实现，用于测试与开发环境；返回固定格式或空字符串。
2. **OpenAISummarizer**：示例性接入 OpenAI 接口，通过 GPT-3.5/4 生成摘要；可配置 model, max_tokens, temperature 等参数。

4.3 扩展指南 / Extension Guide

- **本地模型接入 / Local Model Integration**：编写继承 BaseSummarizer 的类，加载本地 LLM 权重，通过调用推理接口生成摘要。
- **自定义规则引擎 / Rule-Based Engine**：结合关键词提取、TF-IDF 等传统 NLP 方法，生成轻量级摘要，适合无网络环境。
- **异步摘要 / Asynchronous Summarization**：为了提高性能，可在 rotate() 中异步调用摘要器，并在归档完成后统一写入。

第 5 章：存储后端设计

5.1 ActiveStore（短期存储）

- **目标 / Purpose**：快速写入与读取，保存最新对话消息。
- **目前实现 / Current Implementation**：支持 JSON 文件 (store_json.py) 和 SQLite (store_sqlite.py)。
- **关键特性 / Key Features**：
    - 低延迟写入，批量插入优化
    - 可配置存储路径与文件名
    - 简单行级或记录级查询接口

5.2 ArchiveStore（长期存储）

- **目标 / Purpose**：保存已摘要压缩的历史上下文，构建长期对话记忆。

- **目前实现 / Current Implementation**：同样支持 JSON 与 SQLite，可在必要时迁移到更高性能引擎（如 RocksDB）。
- **关键特性 / Key Features**：
    - 支持大规模数据持久化
    - 可按时间、重要性标记过滤检索
    - 提供元数据（时间戳、摘要级别、来源）存储接口

5.3 扩展至 RocksDB / Extension to RocksDB

- **优势 / Advantages**：
    - 高吞吐写入，适合百万／千万级数据
    - 高并发读取，支持多线程访问
    - 内置压缩与合并策略，优化空间利用
- **注意事项 / Considerations**：
    - 写放大与墓碑清理需通过参数调优（CompactOnDeletionCollectorFactory 等）
    - 无跨引擎原子事务，需在应用层保证摘要写入与清理的原子性
    - 异步压缩可能导致读时延迟波动，应监控并设置合适的 compaction 策略

第 6 章：扩展性规划

6.1 存储引擎扩展 / Storage Engine Extensions

- **新增引擎**：除了 JSON、SQLite、RocksDB，还可集成：
    - **Redis**：用于分布式缓存与实时访问
    - **向量数据库**：如 Pinecone、Weaviate，实现基于嵌入的语义检索
    - **对象存储**：AWS S3 或 Ceph，用于冷存储与归档
- **替换策略**：通过实现 StoreInterface 来无缝替换后端，无需修改核心逻辑。

6.2 摘要与标签 / Summarization & Tagging Extensions

- **多模型摘要**：支持并行调用多个摘要模型，融合结果以提高摘要质量
- **主题标签**：引入 TaggerInterface，自动生成话题标签并存储在 ArchiveStore 元数据中
- **情感分析**：通过可插拔情感分析模块，为对话片段添加情感标记

6.3 部署与可观测性 / Deployment & Observability

- **监控指标**：记录以下关键指标并暴露到 Prometheus：
    - Active 写入速度（ops/sec）
    - 归档摘要时延（Latency）
    - Archive 存储大小增长（Bytes）
- **日志记录**：详细记录轮换事件与摘要调用，便于故障排查
- **健康检查**：提供 /health API 接口，检测存储连通性与摘要服务可用性

第 7 章：原创性声明

本文件所述的 **DualDB Memory Plugin** 架构设计于 2025 年 7 月完成，由 **孙旭** 独立完成并首次公开。其核心创新在于：

1. 双数据库分层存储（Active 与 Archive），模拟人类短期/长期记忆的工作机制；
2. 内置多条件自动轮回触发与摘要压缩，简化对话记忆管理流程；
3. 模块化接口设计（存储接口、摘要接口、触发机制），保证可插拔与可扩展性；
4. 轻量级实现，支持 JSON、SQLite 后端，并预留 RocksDB 等高性能存储扩展。
5. 扩展至 RocksDB 与 AI 模型调教的挑战及应对 / Challenges & Mitigations for RocksDB Extension and AI Tuning
    - **跨库一致性风险 / Cross‑Engine Consistency**
        - **挑战**：RocksDB 不支持跨库事务，归档时“写入 Archive → 删除 Active”若出现异常，可能导致数据丢失或重复。
        - **应对**：在应用层引入“待归档队列”与“写入重试”机制，保证写入失败时自动回滚或重试，同时将异常批次记录至监控告警。

- **写放大与墓碑清理 / Write Amplification & Tombstone Management**
    - **挑战**：LCM-Tree 结构在批量删除后会产生大量墓碑并触发写放大，影响 I/O 性能和存储寿命。
    - **应对**：启用并调优 CompactOnDeletionCollectorFactory，定期触发手动或后台 compaction；配置 rate_limiter 限制压缩负载；调整 MemTable 与并行压缩线程参数。
- **高效触发机制 / Efficient Rotation Triggers**
    - **挑战**：基于行扫描判断消息阈值在 RocksDB 中低效，导致触发检测延迟。
    - **应对**：写入时维护原子计数器（消息数、Token 总数等），实时更新触发状态；将关键词与定时触发逻辑保持在应用层执行。
- **并发访问与锁管理 / Concurrency & Locking**
    - **挑战**：多线程或多实例同时执行轮换，可能造成竞态条件或数据不一致。
    - **应对**：利用 RocksDB Snapshot 和 ColumnFamily 特性，或在应用层对 rotate() 操作加互斥锁，确保同一时刻仅有一个轮换任务在运行。
- **AI 模型调用稳定性 / AI Summarizer Stability**
    - **挑战**：摘要服务（OpenAI/Claude/本地模型）接口抖动或超时，可能中断归档流程。
    - **应对**：摘要调用设置超时与重试策略；可选返回“原文+部分摘要”作为降级结果；定期评估摘要时延和质量，动态调整 batch 大小和并发度。

6．本架构设计及其对应代码著作权均归 孙旭 所有，未经许可，不得复制、修改或分发

The copyrights of this architectural design and its corresponding code

belong exclusively to 孙旭; no part may be copied, modified, or redistributed without permission.

All rights reserved.

基于对现有主流对话记忆方案（如 LangChain、Zep、Redis Memory Server 等）的对比分析，未发现与该架构完全同构的实现，故此架构具有结构性原创性。

© 2025 YZXY6151. 本文档及其对应代码的著作权归 YZXY6151 所有，未经许可，不得复制、修改或分发。保留所有权利。